A MINI PROJECT
ON

MAGPIE: A DEMONSTRATION OF END-TO-END ENCRYPTION.

Submitted to

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY, HYDERABAD

In partial fulfilment of the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY
in

COMPUTER SCIENCE AND ENGINEERING
By

AMULYA REDDY Y	**217Y1A0572**
SRINIVAS RAO T	**217Y1A05C0**

Under the Guidance of
Mr. B. T. Srinivas Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

MARRI LAXMAN REDDY
INSTITUTE OF TECHNOLOGY & MANAGEMENT
(AN AUTONOMOUS INSTITUTION)
(Approved by AICTE, New Delhi & Affiliated to JNTUH, Hyderabad)
NAAC Accredited Institution with 'A' Grade & Recognized Under Section2(f) & 12(B)of the UGC act,1956

JULY, 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Date:

CERTIFICATE

This is to certify that the project work entitled "**MAGPIE: A DEMONSTRATION OF END-TO-END ENCRYPTION**" work done by **AMULYA REDDY Y (217Y1A0572), SRINIVAS RAO T (217Y1A05C0) students** of Department of Computer Science and Engineering, is a record of bona fide work carried out by the members during a period from FEB,2024 to JULY,2024 under the supervision of **Mr. B.T. SRINIVAS**. This project is done as a fulfilment of obtaining Bachelor of Technology Degree to be awarded by Jawaharlal Nehru Technological University Hyderabad, Hyderabad.

The matter embodied in this project report has not been submitted by us to any other university for the award of any other degree.

AMULYA REDDY Y
(217Y1A0572)

SRINIVAS RAO T
(217Y1A05C0)

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

Date:

Mr. B.T. SRINIVAS (Professor)

The Viva-Voce Examination of above students, has been held on………………………

External Examiner

Head of the Department

Principal/Director

ACKNOWLEDGEMENTS

We wish to express deepest gratitude and thanks to **Dr. R. Murali Prasad, Principal, and Dr. P. Sridhar, Director** for their constant support and encouragement in providing all the facilities in the college to do the project work.

We are very much grateful to my Project Coordinator, **Dr. T.S Srinivas**, Computer Science and Engineering, MLRITM, Dundigal, Hyderabad, who has not only shown utmost patience, but was fertile in suggestions, vigilant in directions of error and has been infinitely helpful. We are extremely grateful to **Dr. T.S Srinivas**, MLRITM, Dundigal, Hyderabad, for the moral support and encouragement given in completing my project work.

We would like to express my sincere gratitude to my guide **Dr. T. S Srinivas**, Computer Science and Engineering, for his excellent guidance and invaluable support, which helped me accomplish the B.Tech (CSE) degree and prepared me to achieve more life goals in the future. His total support of my dissertation and countless contributions to my technical and professional development made for a truly enjoyable and fruitful experience. Special thanks are dedicated for the discussions we had on almost every working day during my project period and for reviewing my dissertation.

We would also like to thank all our faculties, administrative staff and management of MLRITM, who helped me to complete the mini project.

On a more personal note, I thank my **beloved parents and friends** for their moral support during the course of our project.

INDEX

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The project, named Magpie, is a Python-based application that implements symmetric encryption (END to END Encryption) and hashing techniques to secure text messages. It provides both a command-line interface (CLI) and a graphical user interface (GUI) for users to enter, encrypt, decrypt, and verify text messages using a secret key. The application utilizes the shift cipher for encryption and the `SHA-256` algorithm for hashing.

The main objective of the project is to demonstrate the fundamental principles of cryptography and information security in a simple and interactive manner. It includes a tutorial and a README file that explain the concepts and usage of the application.

Features

- Symmetric encryption using the shift cipher
- Hashing using the SHA-256 algorithm
- Command-line interface (CLI) for text message operations
- Graphical user interface (GUI) for easy interaction
- Secret key-based encryption and decryption
- Text message verification
- Tutorial and README files for guidance

Purpose

The purpose of the Magpie project is to provide a practical example of how symmetric encryption and hashing techniques can be used to secure text messages. By implementing a command-line interface and a graphical user interface, the project aims to make the concepts of cryptography and information security more accessible to users. The tutorial and README files further enhance the understanding of the application's concepts and usage.

Conclusion

Magpie is a Python-based application that showcases the implementation of symmetric encryption and hashing techniques for securing text messages. With its command-line interface, graphical user interface, and comprehensive documentation, the project serves as an educational tool for understanding the basic principles of cryptography and information security.

CHAPTER 1

INTRODUCTION

1.1 PROJECT OVERVIEW

Encryption and decryption are fundamental concepts in the field of cryptography, playing a crucial role in securing sensitive information and communications.

Encryption is the process of converting plaintext (readable data) into ciphertext (unreadable data) using an algorithm and an encryption key. The goal of encryption is to ensure that even if the ciphertext is intercepted or accessed by unauthorized parties, it remains unintelligible without the corresponding decryption key.

Decryption is the process of converting ciphertext back into its original plaintext form using a decryption algorithm and the corresponding decryption key. Decryption reverses the encryption process, allowing authorized parties to access and interpret the original message or data.

Key Components of Encryption and Decryption:

1. **Plaintext:** The original message or data that is to be encrypted.
2. **Ciphertext:** The encrypted form of the plaintext, which appears as random and unintelligible data.
3. **Encryption Algorithm:** A mathematical procedure or algorithm used to transform plaintext into ciphertext. Common encryption algorithms include Advanced Encryption Standard (AES), Data Encryption Standard (DES), and RSA.
4. **Encryption Key:** A piece of information used by the encryption algorithm to perform the encryption process. The security of the encryption relies heavily on the strength of the encryption key.
5.**Decryption Algorithm:** A mathematical procedure or algorithm used to transform ciphertext back into plaintext. The decryption algorithm must be the inverse of the encryption algorithm.
6. **Decryption Key:** A piece of information used by the decryption algorithm to reverse the

encryption process and recover the original plaintext. The decryption key must match the encryption key used to encrypt the data.

In summary, encryption and decryption are essential techniques for securing data and communications, enabling confidentiality, integrity, and authentication in various applications across industries.

1.2 OBJECTIVE

The objective of the Magpie is to provide a simple and secure solution for symmetric encryption and decryption of text messages. The module offers functionalities to generate encryption keys, encrypt plaintext messages, and decrypt ciphertext messages using the Fernet encryption scheme from the cryptography library.

Key Components:

1. Key Management:

   generate_key (): Generates a new symmetric encryption key and saves it to a file named 'key.key'.

   load_key (): Loads the encryption key from the 'key.key' file. If the file does not exist, it generates a new key.

2. Encryption:

   encrypt_message (message: str, key: bytes) -> str: Encrypts a message using symmetric encryption with the provided key.

3. Decryption:

   decrypt_message (encrypted_message: str, key: bytes) -> str | type [Invalid Token] | Exception | type [BinasciiError]: Decrypts an encrypted message using the provided key. It handles potential errors such as invalid tokens or bin ascii errors.

4. Error Handling:

   The module handles errors such as missing key files and invalid tokens during decryption. It raises custom exceptions (`KeyNotFoundError`) to notify users of key-related issues.

5. User Interface:

Magpie aims to offer a user-friendly interface and straightforward implementation, making it accessible to both beginners and experienced users. This could include providing clear documentation, intuitive APIs, and possibly graphical user interfaces (GUIs) or command-line interfaces (CLIs) for interacting with the sentiment analysis functionalities.

Conclusion:

The Symmetric Encryption Module offers a convenient and secure solution for symmetric encryption and decryption tasks in Python applications. It provides an easy-to-use interface while prioritizing security and error handling.

CHAPTER 2

LITERATURE REVIEW

2.1 SURVEY DETAILS

Synthesizing information from scholarly articles, conference papers, books, and technical reports, a literature review on encryption and decryption provides a comprehensive understanding of cryptographic techniques, their applications, and their role in ensuring information security in the digital age.

Encryption and decryption typically cover a broad range of topics, including cryptographic algorithms, security protocols, applications, challenges, and advancements. Here's an overview of what a literature review on encryption and decryption might include:

1. Historical Perspective:

Introduction to the evolution of encryption techniques from ancient times to modern cryptography.

Overview of classical ciphers such as Caesar cipher, Vigenère cipher, and Enigma machine.

2. Fundamental Concepts:

Explanation of basic cryptographic principles like symmetric encryption, asymmetric encryption, hashing, and digital signatures.

Description of common encryption algorithms such as AES (Advanced Encryption Standard), RSA (Rivest-Shamir-Adleman), and ECC (Elliptic Curve Cryptography).

3. Cryptographic Techniques:

Exploration of symmetric encryption algorithms and their modes of operation (e.g., ECB, CBC, GCM).

Discussion of asymmetric encryption algorithms and key exchange protocols (e.g., Diffie-Hellman key exchange, RSA key exchange).

Analysis of hash functions and their applications in data integrity verification.

4. Security Protocols:

Overview of secure communication protocols like TLS (Transport Layer Security) and its predecessor SSL (Secure Sockets Layer).

Examination of secure email protocols such as PGP (Pretty Good Privacy) and S/MIME

(Secure/Multipurpose Internet Mail Extensions).

5. Applications and Use Cases:

Examination of encryption techniques in cloud computing, IoT (Internet of Things), and mobile communication.

Case studies highlighting the role of encryption in protecting sensitive data and mitigating security risks.

6. Security Analysis:

Research papers often include security analysis of cryptographic algorithms and systems, including symmetric encryption schemes like Fernet.

-Security analysis may cover aspects such as resistance to known cryptographic attacks, cryptographic strength, and potential vulnerabilities or weaknesses.

7. Best Practices and Guidelines:

-Literature might include best practices, guidelines, and standards for designing, implementing, and deploying cryptographic systems.

-Best practices may cover areas such as cryptographic algorithm selection, key management, secure coding practices, and compliance with security standards and regulations.

2.2 EXISTING SYSTEM

1. The Existing Version of the Encryption System doesn’t provide the GUI (Graphical User Interface).
2. Haven’t involved Hash libraries (SHA-256) and Fernet.
3. Custom exceptions like key not found error are raised when necessary

2.3 PROPOSED SYSTEM

1. The Proposed Version of the Encryption System provides a User-friendly GUI.
2. Have used Hash libraries (SHA-256) and Fernet Cryptography.
3. Log errors for debugging and auditing purposes

CHAPTER 3

SYSYTEM REQUIREMENTS ANALYSIS

3.1 REQUIREMENT ANALYSIS

Project Perspective:

Magpie implements a Python-based encryption and decryption system using the Fernet symmetric encryption scheme. It offers modular functions for key generation, loading, encryption, and decryption, while incorporating error handling mechanisms. The system interacts with the file system for key management and adheres to security best practices. With documentation, testing, and potential integration into various applications, it ensures secure data communication and storage, emphasizing scalability, compliance, and security auditing for robust deployment.

Project Features:

Magpie features collectively enable the project to offer a reliable and secure encryption and decryption solution, suitable for integration into a range of applications requiring data security.

1. Key management
2. Error Handling
3. Encryption and Decryption
4. Scalability
5. File interaction
6. Robustness
7. Security

3.2 REQUIREMENT SPECIFICATIONS

Certainly, here are the system specifications with numerical values:

1. Operating System:
   - Compatible with Windows, macOS, Linux.
2. Python Version:
   - Requires Python 3.x interpreter.
3. cryptography Library:
   - Utilizes cryptography library.
   - Installation: pip install cryptography.

4. Disk Space:

- Negligible: Minimal disk space for key file and messages.

5. Processing Power:

- Minimal processing power required.

6.RAM:

- Modest: Small amount of RAM needed for script execution and key loading.

7. Input/Output:

- Relies on standard file input/output operations.

8. Network Connectivity:

- Not required for core functionality.

These numerical specifications provide concise information about the system's requirements and capabilities, aiding in deployment and resource allocation.

3.3 LANGUAGE SPECIFICATIONS

Python: Python is a high-level, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation.
Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly procedural), object-oriented and functional programming.
It is often described as a "batteries included" language due to its comprehensive standard library.

3.4 TECHNOLOGIES REQUIRED

This Python code utilizes several libraries and features. Here are the technologies required and used in this code:

1. **Python**: The code itself is written in Python, so you need Python installed on your system to run it.
2. **cryptography**: This library provides cryptographic recipes and primitives to Python developers. In this code, it's used for symmetric encryption and decryption, specifically utilizing "Fernet" which is a symmetric encryption algorithm.
3. **os**: The os module provides a way to interact with the operating system. In this code, it's used for operations like getting the current working directory and checking if a file exists.

4. **os.path**: This module provides functions to interact with the filesystem path. In this code, it’s used for checking if a file exists.
5. **binascii**: This module provides tools for converting binary data to ASCII and vice versa. In this code, it’s used for handling errors related to binary ASCII conversions.
These are the main technologies used in this code. Make sure you have Python installed along with

the cryptography library. You can install cryptography via pip:

```
> Pip install cryptography
```

6. **Tkinter**: Tkinter is the standard GUI (Graphical User Interface) toolkit for Python. It provides a fast and easy way to create GUI applications. It is used for creating windows, buttons, text fields, labels, etc., and handling events like button clicks, text input, etc.
7. **rich**: Rich is a Python library for rich text and beautiful formatting in the terminal. It’s used here

for displaying colorful and formatted text in the command-line interface.

```
> Pip install rich
```

Ensure you have the necessary permissions to perform file operations in the directory where you run this code, as it deals with file I/O for storing and loading keys.

CHAPTER 4

SYSTEM DESIGN

4.1 SYSTEM ARCHITECTURE

Architecture describes the overall functionality of the project with all the modules specified in it. It gives a clear picture of the internal process of the projects. The architecture along with modules can be displayed as the following.

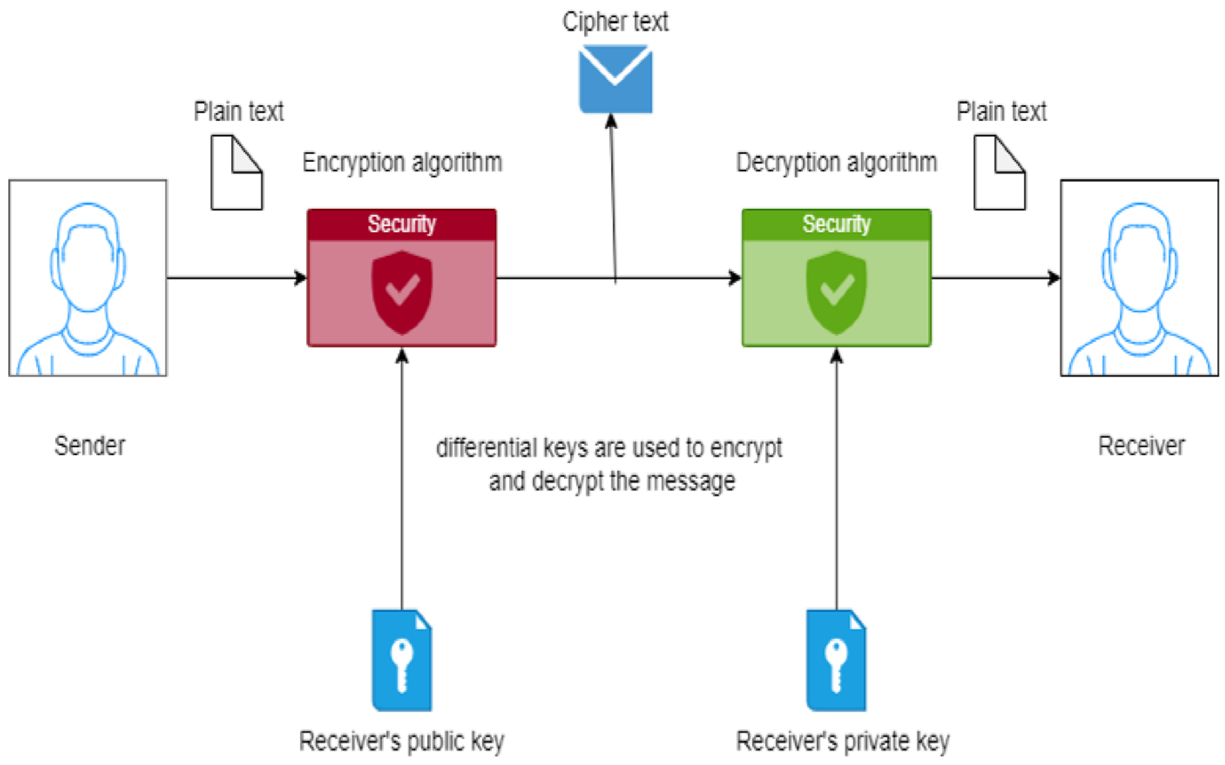


Figure 4.1: Architecture

The above figure shows the basic architecture of the encryption and decryption process.

4.2 USE CASE DIAGRAM

A use case diagram in the Unified Modelling Language (UML) is a type of behavioural diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

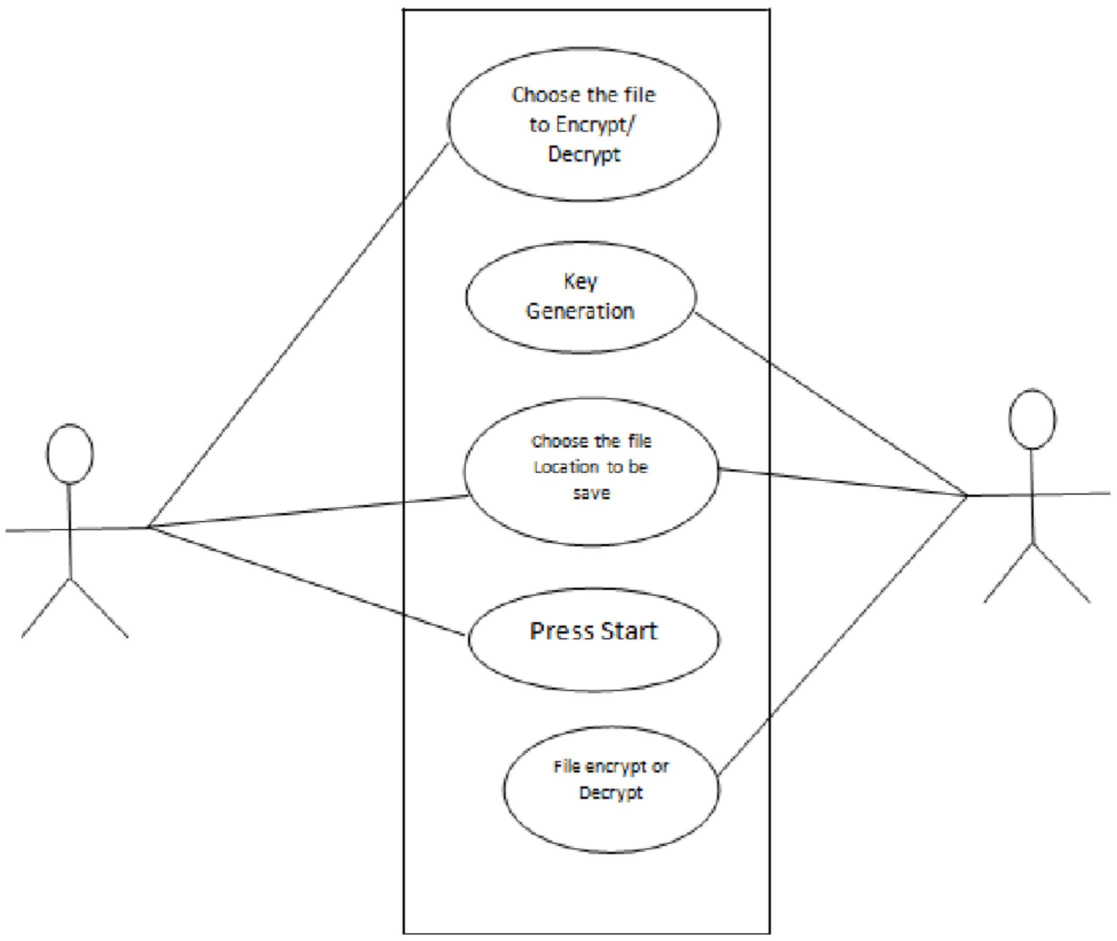

Choose the file to Encrypt/ Decrypt
Key Generation
Choose the file Location to be save
Press Start
File encrypt or Decrypt

Figure 4.2: USE-CASE Diagram

4.3 DATA FLOW DIAGRAM

The DFD is also called as bubble chart. It is a simple graphical format that can be used to represent a system in terms of input data to the system, various processing carried out on this data, and the output data is generated by this system. The data flow diagram (DFD) is one of the most important modelling tools. It is used to model the system components. These components are the system process, the data used by the process, an external entity that interacts with the system and the information flows in the system.

DFD show the information moves through the system and how it is modified by a series of transformations. It is a graphical technique that depicts information flow and the transformations that are applied as data moves from input to output.

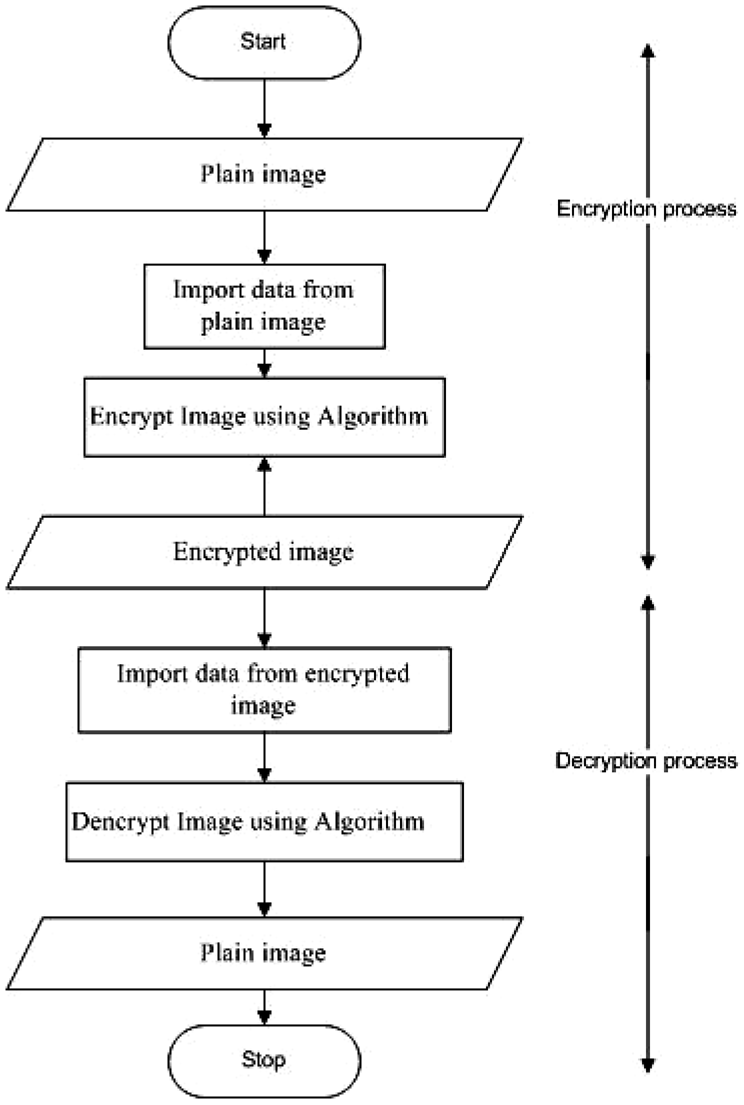

Start
Plain image
Import data from plain image
Encrypt Image using Algorithm
Encrypted image
Import data from encrypted image
Dencrypt Image using Algorithm
Plain image
Stop
Encryption process
Decryption process

Figure 4.3: Data Flow Diagram

4.4 CLASS DIAGRAM

In software engineering, a class diagram in the Unified Modelling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

The class diagram is the main building block of object-oriented modelling. It is used for general conceptual modelling of the structure of the application, and for detailed modelling, translating the models into programming code.

Class diagrams can also be used for data modelling. The classes in a class diagram represent both the main elements, interactions in the application, and the classes to be programmed.

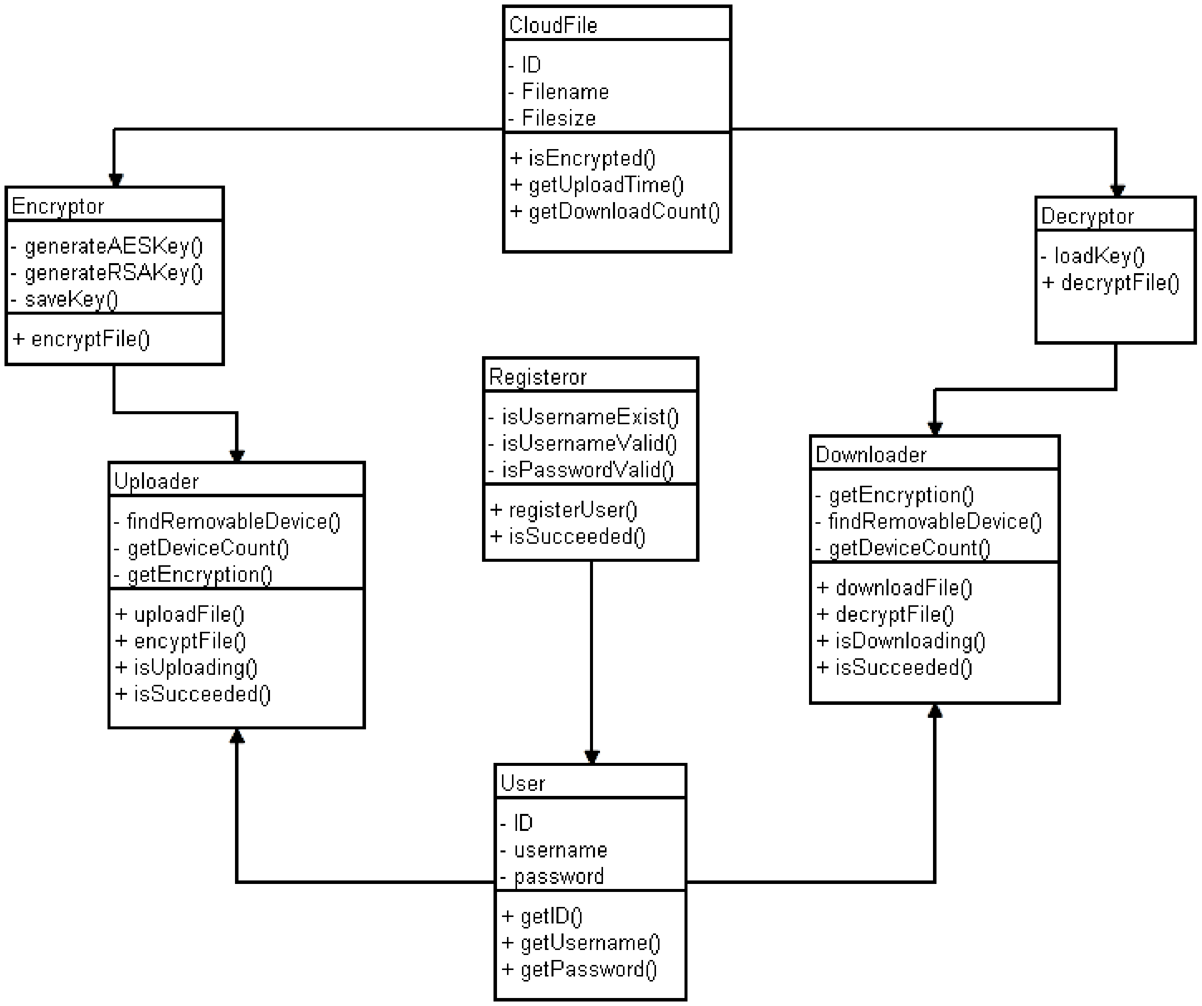


Figure 4.4: Class Diagram

4.5 SEQUENCE DIAGRAM

A sequence diagram is a type of interaction diagram because it describes how—and in what order—a group of objects works together.

These diagrams are used by software developers and business professionals to understand requirements for a new system or to document an existing process.

Sequence diagrams are sometimes known as event diagrams or event scenarios. A sequence diagram shows, as parallel vertical lines (lifelines), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner

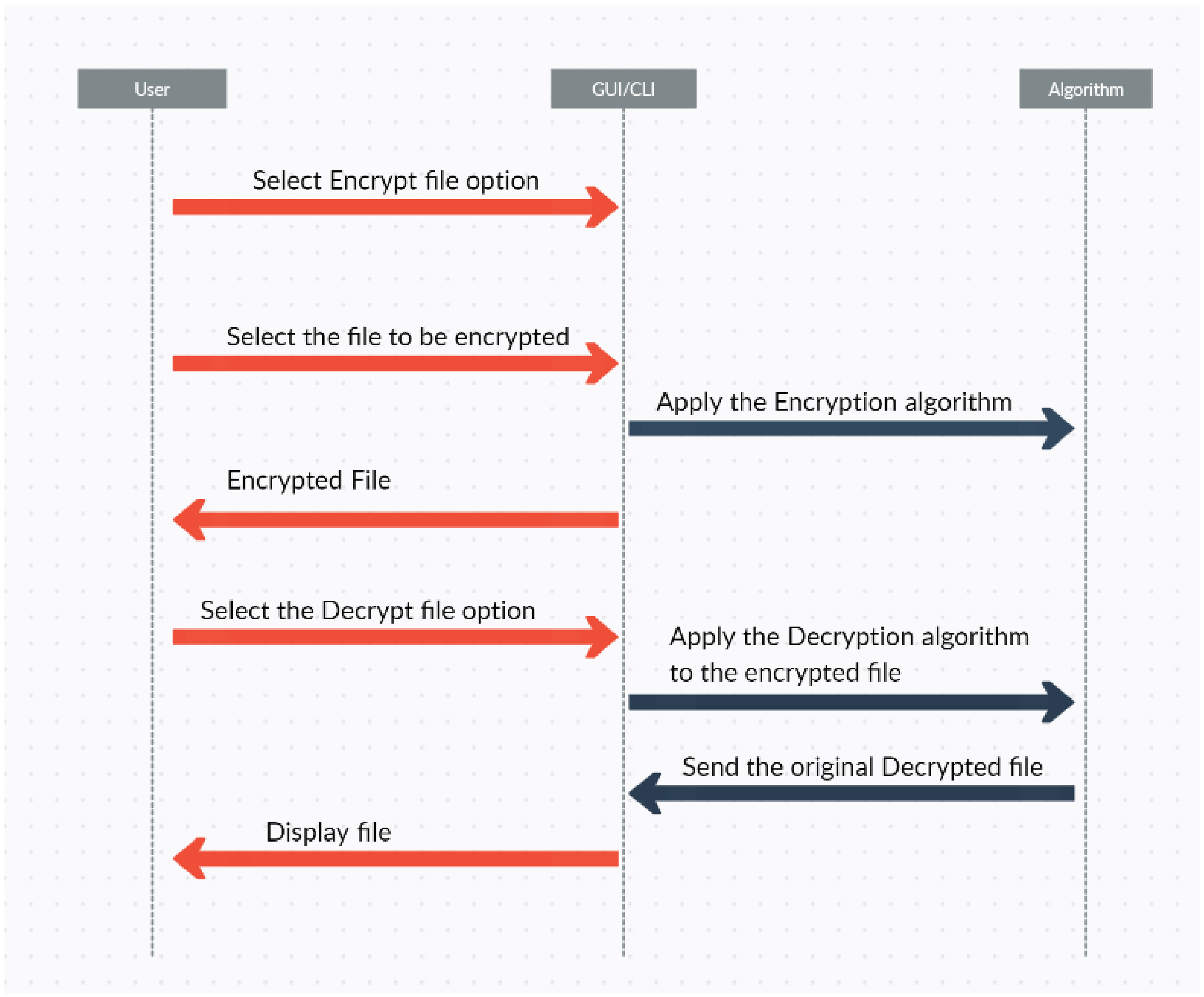

User
GUI/CLI
Algorithm
Select Encrypt file option
Select the file to be encrypted
Apply the Encryption algorithm
Encrypted File
Select the Decrypt file option
Apply the Decryption algorithm to the encrypted file
Send the original Decrypted file
Display file

Figure 4.5: Sequence Diagram

CHAPTER 5

PROJECT DESCRIPTION

The Secure Message Encryption and Decryption System is a Python-based application designed to provide robust encryption and decryption functionalities for securing sensitive messages. Leveraging symmetric encryption with the Fernet algorithm from the `cryptography` library, this system ensures the confidentiality and integrity of messages exchanged between parties.

Key Features:

1. Key Management:

- The system employs a file-based approach for key management, ensuring the security and persistence of encryption keys.
- Upon initialization, the system checks for the presence of a key file in the current directory. If no key file is found, a new key is generated and stored securely.
- Users have the option to manually backup or regenerate keys as needed, enhancing key management flexibility.

2. Encryption:

- Users can encrypt messages of arbitrary length using the provided encryption key.
- The encryption process utilizes the Fernet symmetric encryption algorithm, which guarantees strong security while maintaining efficiency.
- Encrypted messages are returned in a format suitable for transmission over insecure channels, ensuring confidentiality during transit.

3. Decryption:

- Encrypted messages can be decrypted by authorized parties possessing the appropriate decryption key.
- The system validates the integrity of encrypted messages during decryption, detecting any unauthorized modifications or tampering.
- Robust error handling mechanisms are implemented to handle scenarios such as invalid keys or corrupted ciphertexts, ensuring the reliability of the decryption process.

4. Error Handling and Exception Reporting:

- Custom exceptions, including `KeyNotFoundError`, `InvalidToken`, and `BinasciiError`, are

defined to capture and handle specific error conditions gracefully.
- Exception messages provide clear and informative feedback to users, facilitating troubleshooting and resolution of issues encountered during encryption or decryption.

Usage:

1. Encryption Workflow:
- Users initiate the encryption process by providing the plaintext message and the encryption key.
- The system encrypts the message using the specified key and returns the corresponding ciphertext.
- Encrypted messages can then be securely transmitted or stored, safeguarding sensitive information from unauthorized access.

2. Decryption Workflow:
- Authorized recipients obtain encrypted messages and the corresponding decryption key.
- By supplying the encrypted message and the decryption key to the system, users can decrypt the ciphertext and retrieve the original plaintext message.
- Decrypted messages are presented to users in their original form, enabling seamless access to sensitive information while preserving data confidentiality.

Future Enhancements:

1. User Interface Enhancements:
- Develop a user-friendly command-line interface (CLI) or graphical user interface (GUI) to streamline user interactions and improve usability.
- Integrate features such as key generation, encryption, and decryption into a cohesive and intuitive workflow, enhancing user experience.

2. Advanced Encryption Techniques:
- Explore advanced encryption techniques, such as hybrid encryption or authenticated encryption, to enhance security and resilience against emerging threats.

- Investigate the integration of cryptographic protocols like TLS/SSL for secure communication between distributed systems or networked applications.

3. Cross-Platform Compatibility:

- Ensure cross-platform compatibility by optimizing the application for deployment on diverse operating systems, including Windows, macOS, and Linux distributions.
- Implement platform-specific optimizations and compatibility checks to ensure consistent behavior across different environments.

Conclusion:

The Secure Message Encryption and Decryption System represents a fundamental building block for implementing secure communication and data protection in Python-based applications.
By prioritizing key management, robust encryption algorithms, and error handling mechanisms, the system offers a reliable solution for safeguarding sensitive information against unauthorized access and tampering.

CHAPTER 6

IMPLEMENTATION & RESULT

6.1 LIBRARIES USED

1. **cryptography.fernet**: This library provides an implementation of symmetric encryption using the Fernet algorithm. Fernet is a symmetric encryption algorithm that guarantees the integrity and confidentiality of data. It is part of the `cryptography` library, which offers various cryptographic functionalities in Python.

2. **os**: The `os` module provides a way to interact with the operating system. In this script, it's being used to access functionalities related to file handling and directory operations, such as getting the current working directory (`getcwd`) and checking if a file exists (`isfile`).

3. **bin ascii**: The `binascii` module provides functions for converting between binary data and ASCII-encoded hexadecimal representations. In this script, it's specifically importing the `Error` class, which might be used for handling errors related to binary-to-text encoding and decoding.

---These libraries collectively enable the generation, loading, encryption, and decryption of messages using symmetric key cryptography in Python---

1. **tkinter (`import tkinter as tk`):** This library is the standard GUI toolkit for Python. It provides various widgets (such as buttons, labels, text fields, etc.) that can be used to create graphical user interfaces.

2. **ttk (`from tkinter import ttk`)**: The `ttk` module within tkinter provides access to the themed widget set, which includes enhanced versions of the standard tkinter widgets with a more modern and consistent appearance across different platforms.

3. **filedialog (`from tkinter import filedialog`):** The `filedialog` module within tkinter provides dialogs for opening and saving files. It allows users to interactively select files or specify file paths.

---These libraries collectively enable the creation of a GUI application for symmetric encryption and decryption, allowing users to input text, perform encryption or decryption operations, and save/load text to/from files interactively---

1. **symmetric_encryption:** This appears to be a custom module containing functions for generating keys, loading keys, encrypting messages, and decrypting messages using symmetric encryption. It likely contains the implementations of these functionalities using the Fernet algorithm or similar symmetric encryption techniques.

2. **rich (`from rich import print as rprint`):** The `rich` library is used for adding color and style to console output in Python. It provides enhanced features for formatting text, including colors, styles, and panel layouts, making the CLI output more visually appealing and readable.

3. **rich.panel (`from rich.panel import Panel`):** The `Panel` class from the `rich.panel` module is used to create stylized panels for displaying messages or sections of text. It allows for customizing the border style, background colour, and text alignment of the panel, enhancing the presentation of information in the CLI.

4. **rich.console (`from rich.console import Console`):** The `Console` class from the `rich.console` module provides an interface for interacting with the console and controlling the display of text output. It allows for printing styled text, accepting user input, and managing console-related operations in a rich text environment.

-These libraries collectively enable the creation of a command-line tool for symmetric text encryption and decryption, providing users with an interactive interface for encrypting and decrypting messages securely

6.2 CODE OF THE PROJECT

The code of the project comprises of 3 files, `symmetric_encryption.py`, `CLI.py` and `GUI.py`.

`symmetric_encryption.py`

```
from __future__ import annotations
from os import getcwd
from os.path import isfile
from cryptography.fernet import Fernet
from cryptography.fernet import InvalidToken
from binascii import Error as BinasciiError
```

```
class KeyNotFoundError(FileNotFoundError):
    def __init__(self, *args: object) -> None:
        super().__init__(f'''No key found in Dir: '{args[0]}'
New Key file should be generated at same location.''')

    def __str__(self) -> str:
        return super().__str__()


def generate_key() -> bytes:
    key = Fernet.generate_key()
    with open('key.key', 'wb') as key_file:
        key_file.write(key)

    return load_key()


def load_key() -> bytes:
    try:
        if not isfile('./key.key'):
            current_dir = getcwd()
            raise KeyNotFoundError(current_dir)
    except KeyNotFoundError:
        return generate_key()

    with open('key.key', 'rb') as key_file:
        key = key_file.read()
    return key

def encrypt_message(message: str, key: bytes) -> str:
    f: Fernet = Fernet(key)
    encrypted_message = f.encrypt(message.encode())
    return encrypted_message.decode()


def decrypt_message(encrypted_message: str, key: bytes) -> str | type[InvalidToken] |
Exception | type[BinasciiError]:
    f: Fernet = Fernet(key)
    try:
        decrypted_message = f.decrypt(encrypted_message.encode())
    except InvalidToken:
        return InvalidToken
    except BinasciiError:
        return BinasciiError
    except Exception as e:
        return e

    return decrypted_message.decode()
```

./CLI.py

```
from symmetric_encryption import (
    generate_key,
    load_key,
    encrypt_message,
    decrypt_message,
    InvalidToken,
    BinasciiError
)

from rich import print as rprint
from rich.panel import Panel
from rich.console import Console


def cli_chat() -> None:
    console = Console()

    welcome_msg = Panel.fit("[bold yellow]Welcome to the Text encryption and decryption
tool.", border_style="green")
    rprint(welcome_msg)
    rprint("""Here are the options:

        [green]1[/green] Encrypt : [green](e/E)[/green]
        [blue]2[/blue] Decrypt : [blue](d/D)[/blue]
        [red]3[/red] Exit    : [red](b/B)[/red]
          """)

    while True:
        option = console.input('[i bold]Enter option[/i bold]: ')
        match option[0] if option else '':
            case '1' | 'e' | 'E':
                # Text Encrypt
                message = console.input('Enter your [bold green]message[/bold green]: ')
                op = console.input("[#D0B344]Want to generate a new key?[/#D0B344]
(y/n): ")
                if op == 'y':
                    key = generate_key()
                    rprint('Ciphered Text:\n[bold green]' + encrypt_message(message,
key) + '[/bold green]')
                else:
                    key = load_key()
                    rprint('Ciphered Text:\n[bold green]' + encrypt_message(message,
key) + '[/bold green]')

            case '2' | 'd' | 'D':
                # Text Decrypt
                message = console.input('Enter [bold blue]Ciphered Text[/bold blue]: ')
                op = console.input("[#D0B344]Want to enter key?[/#D0B344] (y/n): ")
                if op == 'y':
                    input_key = console.input('Enter your [bold #D0B344]key:[/bold
#D0B344] ')
                    if len(input_key) != 44:  # 32 url-safe base64 to check key length
                        rprint("[red]Check the key and encrypted message. Key must be 32
url-safe base64.[/red]")
                        continue

                    msg = decrypt_message(message, input_key.encode())

                    if msg == InvalidToken:
                        rprint("[red]Invalid Key[/red]")
                    elif msg == BinasciiError:
                        rprint("[red]Invalid Ciphered Text[/red]")
                    else:
                        rprint('Message is:\n[blue]' + str(msg) + '[/blue]')
                else:
                    msg = decrypt_message(message, load_key())

                    if msg == InvalidToken:
                        rprint("[red]Ciphered Text did not match[/red]")
                    elif msg == BinasciiError:
                        rprint("[red]Invalid Bin Text[/red]")
                    else:
                        rprint('Message is:\n[blue]' + str(msg) + '[/blue]')

            case '3' | 'break' | 'b' | 'B' | 'exit' | 'Exit':
                rprint("Thank you :heart:  for using Magpie. Exiting.. :wave:")
                break
            case _:
                rprint("[red]Something's wrong with your input! Please try 
again.[/red]")
```

./GUI.py

```
import tkinter as tk
from tkinter import ttk
from tkinter import filedialog
from symmetric_encryption import generate_key, load_key, encrypt_message, 
decrypt_message, InvalidToken, BinasciiError

ICON_DATA = b''


def browse_files() -> None:
    """Browse for a file and load its content into the top field."""
    key_lable.setvar(key.get())
    try:
        filename = filedialog.askopenfilename(initialdir="/", title="Select a File",
                                              filetypes=(("Text files", "*.txt*"), ("all 
files", "*.*")))

        # Open and read file
        with open(filename, 'r') as file:
            top_text_field.delete('1.0', 'end')
            top_text = file.read()
            top_text_field.insert('1.0', top_text)
    except FileNotFoundError:
        return None


def save_file() -> None:
    """Save the text in the bottom field to a file."""
    filename = filedialog.asksaveasfilename(initialdir="/", title="Save File",
                                            filetypes=(("Text files", "*.txt*"), ("all 
files", "*.*")))
    if filename.partition('.')[0] == '':
        # TODO: Handle empty filename
        return None
    if filename.partition('.')[2] != 'txt':
        # C:\\users\%USERNAME%\downloaded_files\test.txt.csv
        if filename.partition('.')[2] == '':
            filename += '.txt'
        else:
            filename = filename.partition('.')[0] + '.txt'

    # Open and write file
    with open(filename, 'w') as file:
        file.write(bottom_text_field.get('1.0', 'end-1c'))


def convert_text() -> None:
    if radio_bool.get():
        # Complete the encrypt function and also the decrypt function
        if top_text_field.get('1.0', 'end-1c') == '':
            credits_label.config(text='Magpie: No text to encrypt', foreground='red')
            return None

        if key.get() == '':
            get_key()

        _key = key.get()
        plain_text = top_text_field.get('1.0', 'end-1c')
        cipher_text = encrypt_message(plain_text, _key.encode())
        credits_label.config(text='Magpie: Encryption successful', foreground='green')

        bottom_text_field.config(state='normal')  # Enable editing
        bottom_text_field.delete('1.0', 'end')
        bottom_text_field.insert('1.0', cipher_text)
        bottom_text_field.config(state='disabled')  # Disable editing

    else:
        _key = key.get()
        cipher_text = top_text_field.get('1.0', 'end-1c')

        # Extract the decrypted message from the result based on its type
        plain_text = decrypt_message(cipher_text, _key.encode())

        # Handle the decrypted message based on its type
        # Check the type of the decrypted message
        if isinstance(plain_text, str):
            # Insert the decrypted message as a string
            bottom_text_field.config(state='normal')  # Enable editing
            bottom_text_field.delete('1.0', 'end')
            bottom_text_field.insert('1.0', plain_text)
            bottom_text_field.config(state='disabled')  # Disable editing
            credits_label.config(text='Magpie: Decryption successful',
foreground='green')
        elif isinstance(plain_text, InvalidToken):
            credits_label.config(text='Magpie: Invalid Token error occurred',
foreground='red')
        elif isinstance(plain_text, Exception):
            credits_label.config(text='Magpie: Exception occurred during decryption',
foreground='red')
        elif isinstance(plain_text, BinasciiError):
            credits_label.config(text='Magpie: Binascii Error occurred',
foreground='red')
        else:
            credits_label.config(text='Magpie: Unknown error occurred. Please enter
correct Cipher text.',
                                 foreground='red')


def new_key() -> None:
    key.set(generate_key().decode())
    key_lable_text.set('Using KEY:  ' + key.get() + " (Generated)")
    return None


def clear_key() -> None:
    key.set('')


def get_key() -> None:
    key.set(load_key().decode())
    if radio_bool.get():
        key_lable_text.set('Using KEY:  ' + key.get() + " (Loaded)")
    return None


# app
app = tk.Tk()
app.title("Magpie")
app.iconbitmap(r'./magpie_32.ico')
# app.geometry("300x150") # for 730p 'ish screens
app.geometry('685x500')
app.minsize(width=685, height=500)

# VARIABLES
key = tk.StringVar()
radio_bool = tk.BooleanVar(value=True)
key_lable_text: tk.StringVar = tk.StringVar(
    value='Using KEY: ' + key.get() + " (Not yet loaded. Will be loaded on
conversion.)")

# label
greeting_label = ttk.Label(app, text="Welcome to")
greeting_label.pack()

# title
title_label = ttk.Label(app, text='TEXT Encrypter', font='calibre 24 bold')
title_label.pack(side='top')

input_lable = ttk.Label(app, text="Enter your text:")
input_lable.pack(side='top', fill='x', padx=(10, 0))

# top entry field
top_text_field = tk.Text(app, width=50, height=5, background='light blue', wrap='word',
maxundo=15, undo=True, )
```

```
top_text_field.pack(side='top', expand=True, fill='both', padx=10, pady=5)
top_text_field.focus()

# key Frame
key_frame = ttk.Frame(app)

key_lable = ttk.Label(key_frame, textvariable=key_lable_text, font='courier 8 bold')
key_lable.pack(side='left', padx=(10, 0), pady=0)

key_entry = ttk.Entry(key_frame, show=u"\u25CF", width=30, textvariable=key,
foreground='black')

key_entry.configure(validate='key', validatecommand=(key_entry.register(lambda text:
len(text) <= 44), "%P"))

key_load_button: object = ttk.Button(key_frame, text='Load Key', command=get_key,
state='normal')
key_load_button.pack(side='right', padx=10)

key_clear_button: object = ttk.Button(key_frame, text='Generate Key', command=new_key,
state='normal')
key_clear_button.pack(side='right', padx=10)

key_frame.pack(pady=10, side='top', fill='x', anchor='center')

# Options Frame
options_frame = ttk.Frame(app, relief=tk.GROOVE, padding=(10, 5))

# Radio button field
radio_frame = ttk.Frame(options_frame, relief=tk.GROOVE, padding=10)


def radio_func():
    if radio_bool.get():
        get_key()
        key_entry.pack_forget()
        key_clear_button.configure(state='normal', text='Generate Key', command=new_key)

    else:
        key_lable_text.set('Enter your KEY:')
        key_entry.pack(side='left', fill="x", expand=True, padx=10)
        key.set('')
        key_clear_button.configure(text='Clear Key', command=clear_key)


# options
encrypt_radio = (ttk.Radiobutton(radio_frame,
                                 text='Encrypt',
                                 value=True,
                                 command=radio_func,
                                 variable=radio_bool)
                 .pack(side='left', fill='y', padx=20))

decrypt_radio = (ttk.Radiobutton(radio_frame,
                                 text='Decrypt',
                                 value=False,
                                  command=radio_func,
                                  variable=radio_bool)
                 .pack(side='left', fill='y', padx=20))

radio_frame.pack(side='left')

# convert button
convert_button = ttk.Button(options_frame, text='Convert', command=convert_text)
convert_button.pack(side='left', padx=(10, 10))

options_frame.pack(side='top', pady=10)

file_frame = ttk.Frame(options_frame, relief=tk.GROOVE, padding=5)

open_file_button = (ttk.Button(file_frame, text='Open File', command=browse_files).
                    pack(side="top", pady=5))
save_file_button = (ttk.Button(file_frame, text='Save Text', command=save_file).
                    pack(side="top", pady=5))

file_frame.pack(expand=True, fill='x', pady=10, side='right')

output_lable = (ttk.Label(app, text="Your Output:").
                pack(side='top', fill='x', padx=(10, 0)))
# Output field

# bottom entry field
bottom_text_field = tk.Text(app, width=50, height=5, background='light yellow',
wrap='word')
bottom_text_field.pack(side='top', expand=True, fill='both', padx=10, pady=5)

credits_label = ttk.Label(app, text="Made by: Magpie", font='calibre 10 bold')
credits_label.pack(side='bottom', pady=10)

# Start the main event loop
app.mainloop()
```

6.3 RESULT / OUTPUT SCREENS

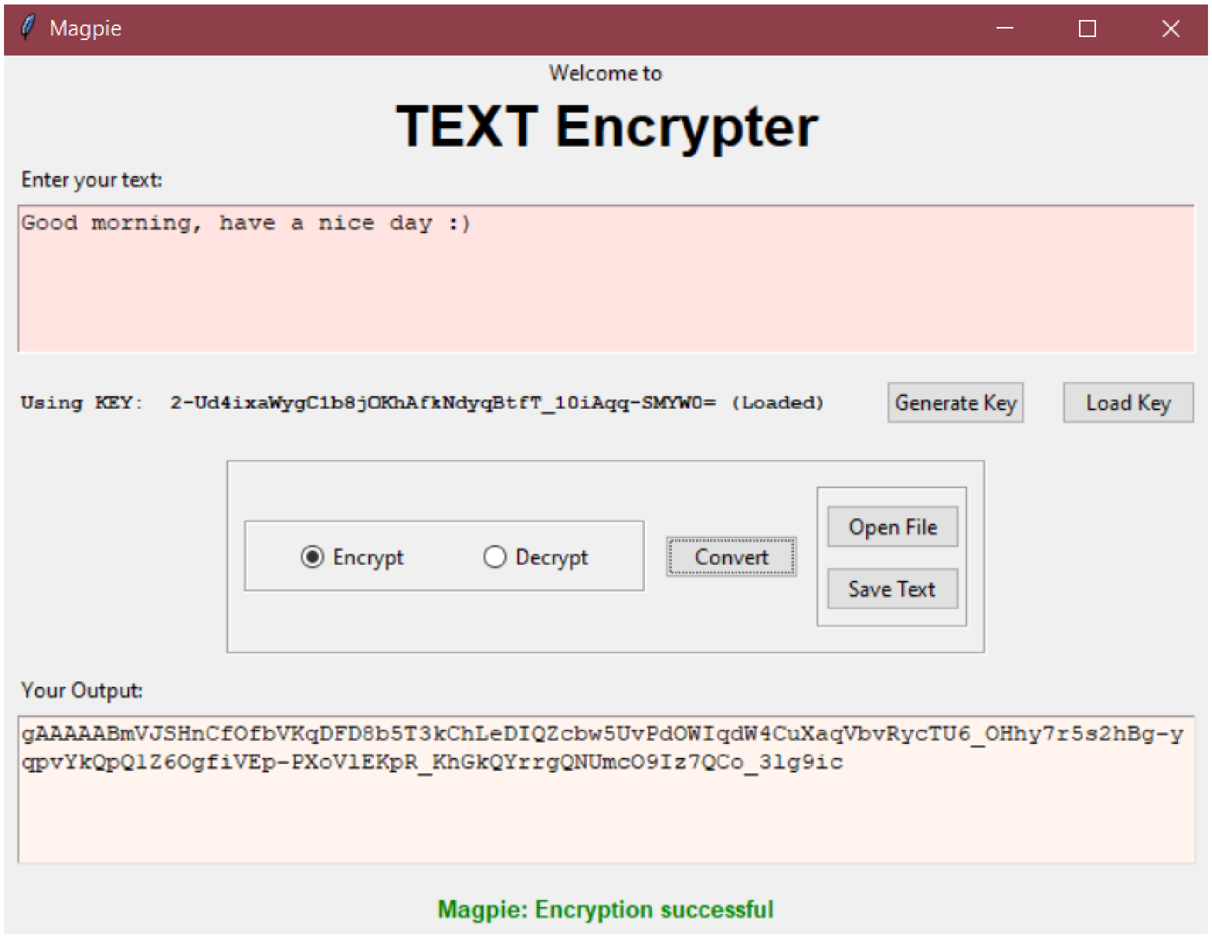
To encrypt-decrypt test files using the GUI tool

1. Run the Magpie installer to install the necessary dependencies.
2. Click the Open File button to choose a text file.
3. Enter the key in the corresponding field.
4. Choose one of the following options:

- ○ Encrypt a message: Select the Encrypt button.
- ○ Decrypt a message: Select the Decrypt button.
- ○ Exit the tool: Click the X button.

5. Click on the Convert button to start the encryption or decryption process.
6. Save the encrypted or decrypted text file to the desired location.
7. Follow the prompts to enter the necessary information for encryption or decryption.

OUTPUT:



```
Good morning, have a nice day :)
```

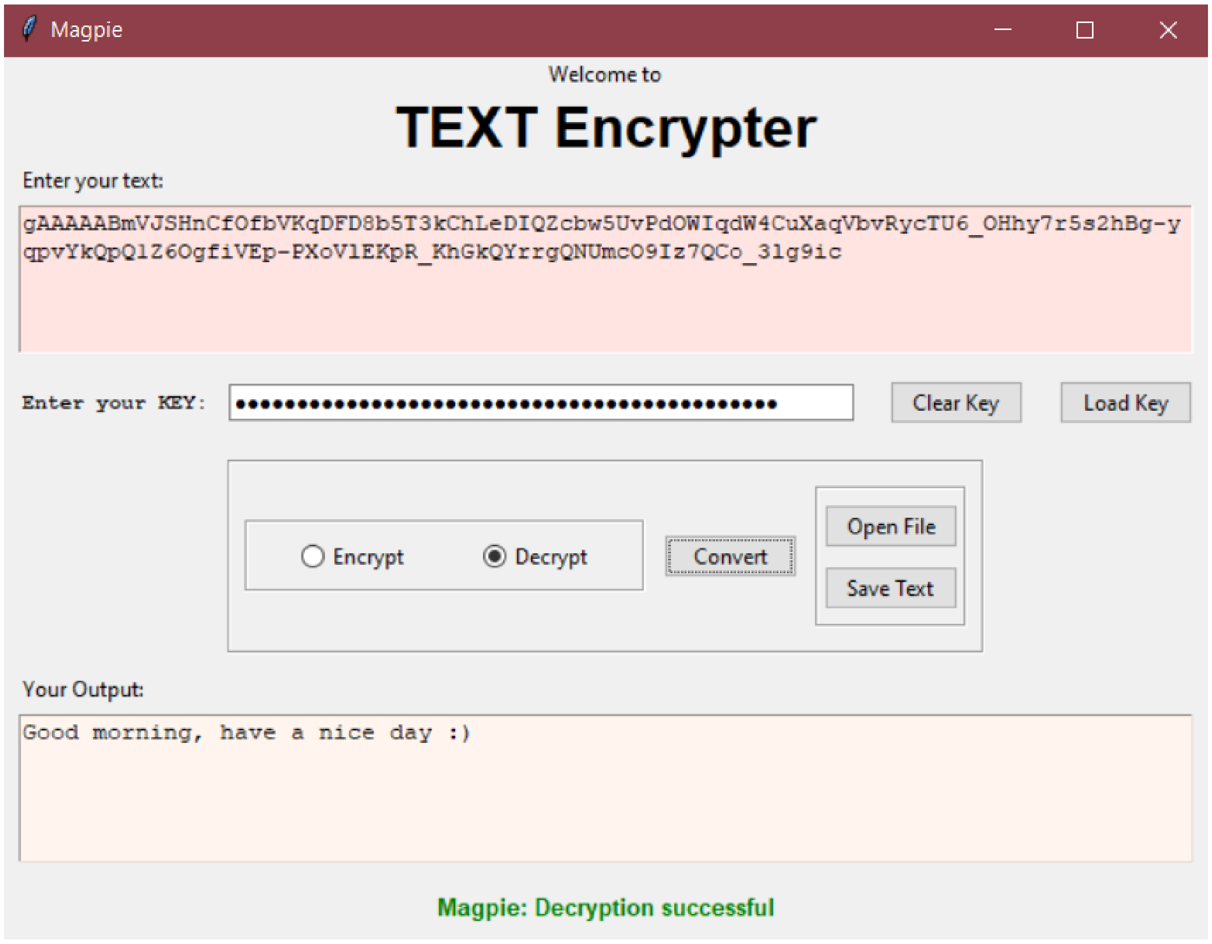
```
Using KEY:  2-Ud4ixaWygC1b8jOKhAfkNdyqBtfT_10iAqq-SMYW0= (Loaded)
```

Encrypt: ● Decrypt: ○

```
gAAAAABmVJSHnCfOfbVKqDFD8b5T3kChLeDIQZcbw5UvPdOWIqdW4CuXaqVbvRycTU6_OHhy7r5s2hBg-y
qpvYkQpQ1Z6OgfiVEp-PXoV1EKpR_KhGkQYrrgQNUmcO9Iz7QCo_3lg9ic
```

Magpie: Encryption successful

Figure 6.2.1: Plain Text to Cipher text (Encryption)



```
gAAAAABmVJSHnCfOfbVKqDFD8b5T3kChLeDIQZcbw5UvPdOWIqdW4CuXaqVbvRycTU6_OHhy7r5s2hBg-y
qpvYkQpQ1Z6OgfiVEp-PXoV1EKpR_KhGkQYrrgQNUmcO9Iz7QCo_31g9ic
```

Enter your KEY: ••

Encrypt [✓ Decrypt]

Your Output:

```
Good morning, have a nice day :)
```

Magpie: Decryption successful

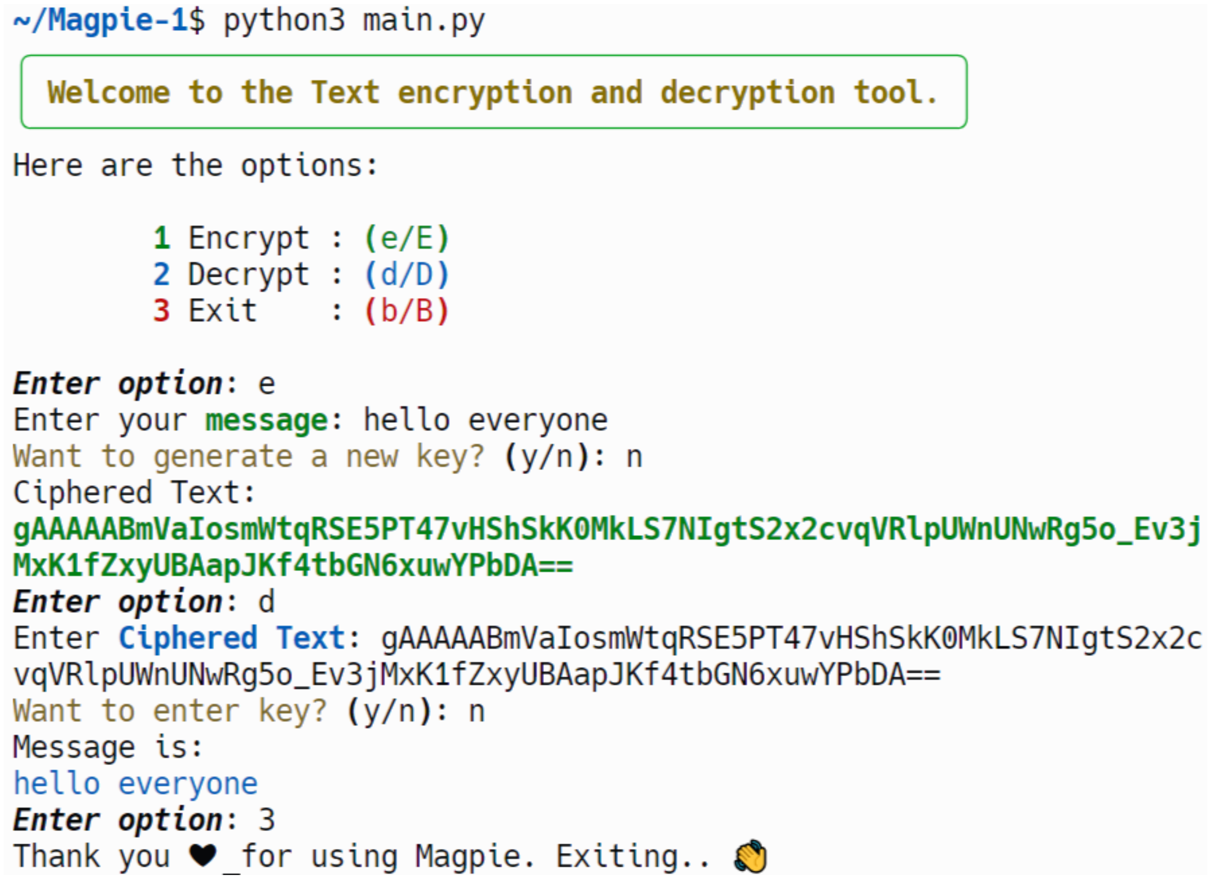
Figure 6.2.2: Cipher text to Plain text (Decryption)

To encrypt-decrypt test files using the CLI tool

1. Run the script symmetric_encryption.py to import the necessary functions.
2. Execute the main script text_encryption.py to start the tool.
3. Choose one of the following options:
   - Encrypt a message: Enter 1, e, or E.
   - Decrypt a message: Enter 2, d, or D.
   - Exit the tool: Enter 3, break, b, B, exit, or Exit.
4. Follow the prompts to enter the necessary information for encryption or decryption.

5. The tool will display the encrypted or decrypted message based on the chosen option.

OUTPUT:



```
~/Magpie-1$ python3 main.py

 Welcome to the Text encryption and decryption tool.

Here are the options:

        1 Encrypt : (e/E)
        2 Decrypt : (d/D)
        3 Exit    : (b/B)

Enter option: e
Enter your message: hello everyone
Want to generate a new key? (y/n): n
Ciphered Text:
gAAAAABmVaIosmWtqRSE5PT47vHShSkK0MkLS7NIgtS2x2cvqVRlpUWnUNwRg5o_Ev3j
MxK1fZxyUBAapJKf4tbGN6xuwYPbDA==
Enter option: d
Enter Ciphered Text: gAAAAABmVaIosmWtqRSE5PT47vHShSkK0MkLS7NIgtS2x2c
vqVRlpUWnUNwRg5o_Ev3jMxK1fZxyUBAapJKf4tbGN6xuwYPbDA==
Want to enter key? (y/n): n
Message is:
hello everyone
Enter option: 3
Thank you ❤_for using Magpie. Exiting.. 👋
```

Figure 6.2.3: Encryption and Decryption using CLI

CHAPTER 7

TESTING

7.1 TESTING STRATEGIES

The purpose of testing is to discover errors. Testing is the process of trying to discover very conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

White Box Testing

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

Black Box Testing

Black Box Testing is testing the software without any knowledge of the inner Workings, structure or language of the module being tested.

Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document.

It is a testing in which the software under test is treated, as a black box. You cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

7.2 TYPES OF TESTING

Unit testing

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated.

Integration testing

Integration tests are designed to test integrated software components to determine if they run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of

Functional Testing

Functional tests provide systematic demonstrations functions tested are available as that specified by the business and technical requirements, system documentation, and user manuals. Functional testing is centered on the following items:

Valid Input: identified classes of valid input must be accepted.
Invalid Input: identified classes of invalid input must be rejected.
Functions: identified functions must be exercised.
Output: identified classes of application outputs must be exercised.
Systems / Procedures: interfacing systems or procedures must be invoked.

Performance Testing

Assess the system's performance under various conditions, including processing speed, resource utilization, and scalability to handle varying workloads.

Robustness Testing

Test the system's resilience to input variations, such as changes in lighting conditions, different writing instruments, and alterations in signature styles.

Usability Testing

Evaluate the user interface and user experience to ensure that the forgery detection system is user-friendly and accessible for its intended users.

Security Testing

Verify the system's resistance to adversarial attacks and attempts to manipulate the forgery detection process, ensuring the security of sensitive signature data.

Cross-browser and Cross-platform Testing

Confirm that the forgery detection system works consistently across different web browsers and platforms, ensuring a broad user reach.

Scalability Testing

Assess the system's ability to handle an increasing volume of signatures and users, ensuring it can scale effectively without compromising performance.

Regression Testing

Perform tests after each system update or modification to ensure that new changes do not negatively impact existing functionality.

Accuracy Testing

Validate the accuracy of the forgery detection system by comparing its results against a trusted benchmark, possibly using a curated dataset with known genuine and forged signatures.

User Acceptance Testing (UAT)

Involve end-users in the testing process to gather feedback on the system's usability, functionality, and overall performance.

Documentation Testing

Verify that documentation, including user manuals and system guides, accurately reflects the system's features and provides comprehensive instructions.

Continuous Testing

Implement automated testing processes to ensure continuous monitoring and rapid detection of issues during the development life cycle.

Ethical Considerations Testing

Assess the forgery detection system for potential biases and ethical concerns, ensuring fairness and equity in its application.

7.3 TEST CASES

Writing Test Case

Test Case 1: Empty Message

Description: Test the behaviour when an empty message is encrypted and decrypted.

Test Steps:

- Encrypt an empty message.
- Decrypt the encrypted empty message.
- Verify if the decrypted message is also empty.

Test Case 2: Large Message

Description: Test the behaviour when a large message is encrypted and decrypted.

Test Steps:

- o Encrypt a large message (larger than the block size).
- o Decrypt the encrypted large message.
- o Verify if the decrypted large message matches the original large message.

Test Case 3: Binary Message

Description: Test the behaviour when a binary message is encrypted and decrypted.

Test Steps:

- o Encrypt a binary message.
- o Decrypt the encrypted binary message.
- o Verify if the decrypted binary message matches the original binary message.

Test Case 4: Encrypt and Decrypt a Message

Description: Encrypt a message, then decrypt it and verify if the decrypted message matches the original message.

Test Steps:

- o Encrypt a sample message using the encrypt_message function.
- o Decrypt the encrypted message using the decrypt_message function.
- o Verify if the decrypted message matches the original message.

Test Case 5: Key File Not Found

Description: Test the behaviour when the key file is not found.

Test Steps:

- o Remove the key.key file from the directory.
- o Attempt to encrypt or decrypt a message.
- o Verify if the function generates a new key file and performs the encryption or decryption.

Test Case 6: Invalid Key

Description: Test the behaviour when an invalid key is provided for decryption.

Test Steps:

- Encrypt a sample message.
- Attempt to decrypt the encrypted message using an invalid key.
- Verify if the function raises an InvalidToken exception.

CHAPTER 8

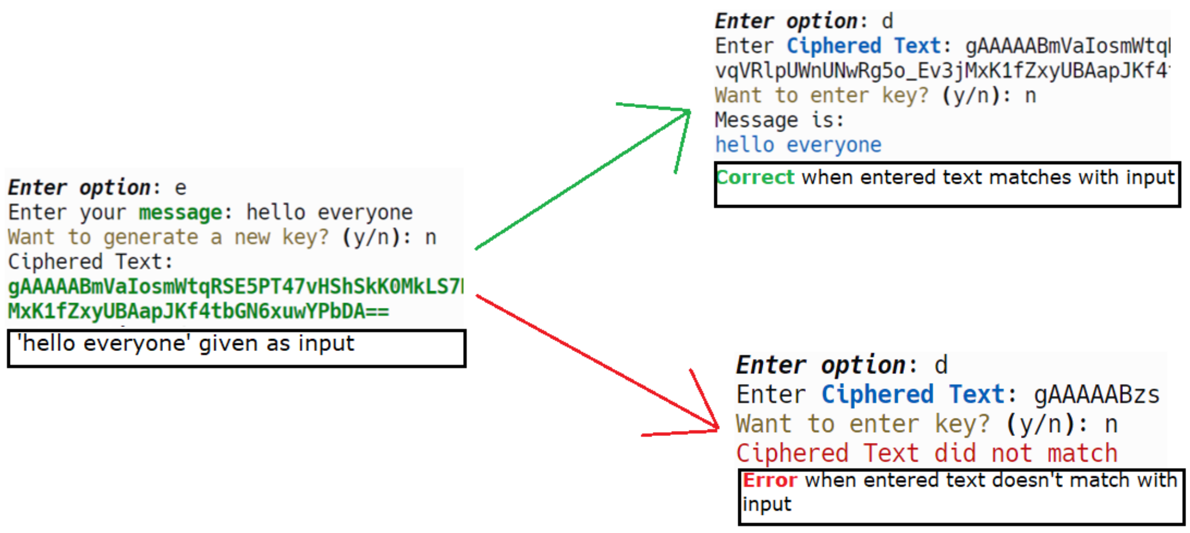
SYSTEM RESULTS

8.1 OUTCOMES OF THE PROPOSED SYSTEM

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases test strategy and approach. Field testing will be performed manually, and functional tests will be written in detail.

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects. The task of the integration test is to check that components or software applications, example: components in a software system or one step up software applications at the company level interact without error. Test Results All the test cases mentioned above passed successfully. No defects encountered. User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements. Test Results: All the test cases mentioned above passed successfully. No defects encountered.

8.2 SCREENSHOTS



```
Enter option: e
Enter your message: hello everyone
Want to generate a new key? (y/n): n
Ciphered Text:
gAAAAABmVaIosmWtqRSE5PT47vHShSkK0MkLS7
MxK1fZxyUBAapJKf4tbGN6xuwYPbDA==
```

'hello everyone' given as input

```
Enter option: d
Enter Ciphered Text: gAAAAABmVaIosmWtq
vqVRlpUWnUNwRg5o_Ev3jMxK1fZxyUBAapJKf4
Want to enter key? (y/n): n
Message is:
hello everyone
```

Correct when entered text matches with input

```
Enter option: d
Enter Ciphered Text: gAAAAABzs
Want to enter key? (y/n): n
Ciphered Text did not match
```

Error when entered text doesn't match with input

Figure 8.1: final result

CHAPTER 9

CONCLUSION AND FUTURE SCOPE

9.1 CONCLUSION

The encryption and decryption mechanism encapsulated within the provided code represents a cornerstone of modern data security practices. By leveraging symmetric encryption, wherein the same key is used for both encryption and decryption, the mechanism ensures a streamlined and efficient approach to securing sensitive information. The Fernet module, a part of the widely-used cryptography library, serves as the backbone of this mechanism, offering a robust and battle-tested encryption algorithm. Encryption, the process of transforming plaintext into ciphertext, safeguards data against unauthorized access during transmission or storage. This is accomplished through the encrypt message function, which accepts a plaintext message and an encryption key, facilitating the conversion of the message into an encrypted format. Conversely, decryption, the reverse process of encryption, is facilitated by the decrypt message function. Here, the encrypted message is decrypted using the same encryption key, ensuring the recovery of the original plaintext. The mechanism further prioritizes resilience through meticulous error handling, ensuring that potential decryption errors, such as invalid tokens or formatting inconsistencies, are promptly addressed. Additionally, robust key management practices are enforced to maintain the confidentiality and integrity of the encryption process. The generate key function generates a new encryption key and persists it in a file, while the load key function ensures consistent access to the encryption key for subsequent encryption and decryption operations. This comprehensive approach to encryption and decryption not only safeguards data against unauthorized access but also helps in gaining confidence in the security and reliability of sensitive information across diverse applications and environments.

9.2 FUTURE SCOPE

The encryption and decryption mechanism presented in the provided code already offers a solid foundation for securing sensitive data. However, there are several avenues for future enhancement and expansion:

Multi-factor Authentication: Enhancing the encryption mechanism to support multi-factor authentication can add an extra layer of security. This could involve incorporating biometric data, token-based authentication, or other factors beyond traditional passwords.

REFERENCES

symmetric_encryption.py

1. **Cryptography Library:**
   - *Cryptography*: A package designed to expose cryptographic recipes and primitives to Python developers. It includes both high-level recipes and low-level interfaces to common cryptographic algorithms such as symmetric ciphers, message digests, and key derivation functions.
     - Cryptography. (n.d.). Retrieved from https://cryptography.io/
2. **Python Standard Library:**
   - *os Module*: This module provides a way of using operating system-dependent functionality like reading or writing to the file system.
     - Python Software Foundation. (n.d.). os — Miscellaneous operating system interfaces. In *Python 3 Documentation*. Retrieved from https://docs.python.org/3/library/os.html
   - *os.path Module*: This module implements some useful functions on pathnames.
     - Python Software Foundation. (n.d.). os.path — Common pathname manipulations. In *Python 3 Documentation*. Retrieved from https://docs.python.org/3/library/os.path.html
   - *binascii Module*: This module contains a number of methods to convert between binary and various ASCII-encoded binary representations.
     - Python Software Foundation. (n.d.). binascii — Convert between binary and ASCII. In *Python 3 Documentation*. Retrieved from https://docs.python.org/3/library/binascii.html
3. **PEP 484 – Type Hints:**
   - *Type Annotations*: Python's type annotations enable optional type checking via static type checkers, which can find type errors before the code runs.
     - van Rossum, G., & Lehtosalo, J. (2014). PEP 484 – Type Hints. Python Enhancement Proposals. Retrieved from https://peps.python.org/pep-0484/

Here are the references for the two Python scripts:

CLI.py

4. **Rich Library Documentation**: The Rich library is used for creating attractive command-line interfaces. The print function from rich and the Panel, Console classes are utilized for displaying text in various styles. You can refer to the official documentation for more details:
   - Rich Documentation
5. **Python Match Statement**: The match statement in Python, introduced in version 3.10, is used for pattern matching. It helps in selecting code blocks to execute based on the structure of the input. Official documentation is available at:
   - Python Pattern Matching
6. **Python File I/O**: The script utilizes basic Python file I/O operations to load and save keys and messages. This includes opening, reading, and writing files. Documentation can be found here:
   - Python File I/O

GUI.py

7. **Tkinter Library Documentation**: Tkinter is the standard Python interface to the Tk GUI toolkit. It is used extensively in this script for creating the GUI elements like buttons, labels, text fields, etc. The documentation provides a comprehensive guide to the various components:
   - Tkinter Documentation
8. **Tkinter filedialog Module**: The filedialog module is part of Tkinter and is used to open and save files through a file dialog box in the GUI. The official reference is available at:
   - Tkinter filedialog
9. **Python StringVar and BooleanVar**: The script uses StringVar and BooleanVar from Tkinter to manage the state of variables that are tied to widgets in the GUI. These are commonly used for managing user input and control states. Documentation is here:
   - StringVar and BooleanVar
10. **Base64 Encoding**: The script checks the length of keys and utilizes Base64 encoding for secure message encryption. Python's base64 module handles this encoding:
    - Base64 Encoding in Python